

PAPER • OPEN ACCESS

## Differentiable programming for online training of a neural artificial viscosity function within a staggered grid Lagrangian hydrodynamics scheme

To cite this article: Pake Melland *et al* 2021 *Mach. Learn.: Sci. Technol.* **2** 025015

View the [article online](#) for updates and enhancements.

You may also like

- [Random phase screen influence of the inhomogeneous tissue layer on the generation of acoustic vortices](#)  
Zhiyao Ma, , Jun Ma et al.
- [Multiple off-axis acoustic vortices generated by dual coaxial vortex beams](#)  
Wen Li, , Si-Jie Dai et al.
- [A novel curve fitting method for AV optimisation of biventricular pacemakers](#)  
Hakim-Moulay Dehbi, Siana Jones, S M Afzal Sohaib et al.



## PAPER

## OPEN ACCESS




RECEIVED  
27 August 2020REVISED  
6 December 2020ACCEPTED FOR PUBLICATION  
23 December 2020PUBLISHED  
26 February 2021

Original Content from  
this work may be used  
under the terms of the  
[Creative Commons  
Attribution 4.0 licence](#).

Any further distribution  
of this work must  
maintain attribution to  
the author(s) and the title  
of the work, journal  
citation and DOI.



# Differentiable programming for online training of a neural artificial viscosity function within a staggered grid Lagrangian hydrodynamics scheme

Pake Melland<sup>1,5</sup> , Jason Albright<sup>2</sup>  and Nathan M Urban<sup>3,4</sup> <sup>1</sup> Applied Mathematical and Computational Sciences, University of Iowa, Iowa City, IA 52242, United States of America<sup>2</sup> Applied Physics, Los Alamos National Laboratory, Los Alamos, NM 87544, United States of America<sup>3</sup> Applied Mathematics, Brookhaven National Laboratory, Upton, NY 11973, United States of America<sup>4</sup> Computational Physics and Methods, Los Alamos National Laboratory, Los Alamos, NM 87544, United States of America<sup>5</sup> NSF-funded guest scientist at Los Alamos National Laboratory during summer 2019.E-mail: [pake-hagen@uiowa.edu](mailto:pake-hagen@uiowa.edu)

Keywords: differentiable programming, Lagrangian hydrodynamics, hybrid modeling, shock waves

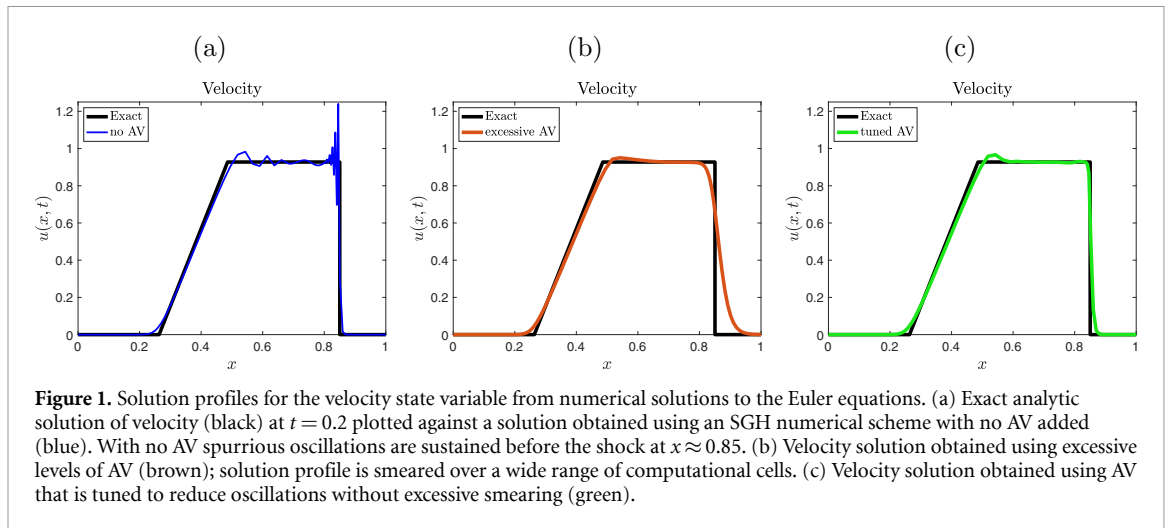
## Abstract

Lagrangian methods to solve the inviscid Euler equations produce numerical oscillations near shock waves. A common approach to reducing these oscillations is to add artificial viscosity (AV) to the discrete equations. The AV term acts as a dissipative mechanism that attenuates oscillations by smearing the shock across a finite number of computational cells. However, AV introduces several control parameters that are not determined by the underlying physical model, and hence, in practice are tuned to the characteristics of a given problem. We seek to improve the standard quadratic-linear AV form by replacing it with a learned neural function that reduces oscillations relative to exact solutions of the Euler equations, resulting in a hybrid numerical-neural hydrodynamic solver. Because AV is an artificial construct that exists solely to improve the numerical properties of a hydrodynamic code, there is no offline ‘viscosity data’ against which a neural network can be trained before inserting into a numerical simulation, thus requiring online training. We achieve this via differentiable programming, i.e. end-to-end backpropagation or adjoint solution through both the neural and differential equation code, using automatic differentiation of the hybrid code in the Julia programming language to calculate the necessary loss function gradients. A novel offline pre-training step accelerates training by initializing the neural network to the default numerical AV scheme, which can be learned rapidly by space-filling sampling over the AV input space. We find that online training over early time steps of simulation is sufficient to learn a neural AV function that reduces numerical oscillations in long-term hydrodynamic shock simulations. These results offer an early proof-of-principle that online differentiable training of hybrid numerical schemes with novel neural network components can improve certain performance aspects existing in purely numerical schemes.

## 1. Introduction

The Euler equations are a system of non-linear hyperbolic partial differential equations (PDEs) derived from conservation laws describing the dynamics of a compressible material, for example gases and liquids, at high pressures, for which the effects of viscosity are neglected [1]. When formulated with Riemann-type initial conditions, initial conditions that are piecewise constant with a single discontinuity, solutions to the Euler equations admit shock, contact, and rarefaction waves [2]. Developing numerical methods that can capture solution profiles exhibiting these low-regularity features is essential in many applications.

Most standard numerical schemes for hyperbolic conservation laws without special modifications introduce spurious, nonphysical oscillations that occur near shocks. These oscillations are a numerical artifact caused by the lack of a mechanism in the Euler equation to convert kinetic energy into internal



energy, and thereby a mechanism to generate the required entropy increase across shocks. Hence, one approach to suppressing these oscillations relies on augmenting the discrete equations with a mechanism to increase entropy across shocks.

Specifically, in the context of a staggered-grid Lagrangian hydrodynamics (SGH) scheme considered here spurious oscillations can develop near shock waves where smoothness assumptions break down (figure 1(a)). Hence, in order to reduce the oscillations, a fictitious term called artificial viscosity (AV) is added to the pressure. This approach goes back to the work done by VonNeumann and Richtmyer in [3]. AV fills the role of the missing dissipative mechanism in the discrete inviscid flow equations and is able to reduce oscillations near shocks at the cost of smearing the shock profile across several computational cells (figures 1(a) and (c)).

We emphasize that while the addition of AV into an SGH scheme is well-founded, it is not derived from the conservation laws underlying the Euler equations; AV is a term that is added to improve numerical aspects of the solution and introduces control parameters that determine its magnitude and other properties. In particular, the well-known linear-quadratic form of AV, see [4–6], introduces two free parameters to control its magnitude. Figure 1 illustrates the effects of augmenting pressure with different levels of AV. Figure 1(a) highlights the profile of a velocity solution obtained with no AV added—spurious oscillations form before the shock. Conversely figure 1(b) shows a solution obtained with excessive AV—the shock profile is smeared over several cells. Figure 1(c) shows the velocity solution obtained by tuning parameters [7] in a traditional AV function to reduce oscillations without excessive smearing of the shock profile. A source of concern for traditional approaches is that the process of tuning parameters in an AV model can be immediately limited by the functional form specified for AV, e.g. quadratic or linear-quadratic. Therefore, the goal of this study is to replace the linear-quadratic form of AV in a Lagrangian staggered grid scheme with a learnable function, with a neural representation in the form of an artificial neural network (ANN), trained to reduce oscillations, resulting in a hybrid numerical-neural scheme.

### 1.1. Computational fluid dynamics and machine learning

Machine learning algorithms are data-driven and rely on patterns in data rather than explicit computational instructions. On the other hand, the algorithms used in scientific computing have their roots in solving explicit, mathematical models typically derived from first principles describing the underlying physics of a problem. In particular, algorithms in computational fluid dynamics (CFD) historically have taken this approach to modeling and rely on approximating solutions to the physically derived Navier–Stokes equations or the inviscid Euler equations using a variety of different numerical methods. The approach taken in this work seeks to build on the robustness of existing numerical methods, by replacing one component, namely AV as part of an existing SGH scheme (described in section 2.2), with an ANN. This approach fits into emerging, but rapidly expanding, work that attempts to combine elements of machine learning with scientific computing. We note that this general approach is not limited to the field of CFD. For example the authors in [8] develop the methodology of universal differential equations for effectively enhancing scientific models with machine learning structures. In the following we review approaches that employ data-driven components to CFD and PDE models.

Work done in [9] and [10] uses convolutional neural networks (CNNs) to improve computational efficiency aspects of fluid simulations. In [10] the CNN is trained on fluid simulation velocity fields and subsequently used as a generative model to produce efficient fluid simulations independent of existing

solvers. In [9] the CNN architecture is used to replace a linear solver for a projection step inside of an Eulerian scheme for solving the inviscid Euler equations resulting in a hybrid model designed to improve run time by replacing the computationally intensive projection step. Similar to [9] the work done in [11] uses ANNs to simulate higher dimensional PDEs by building a mesh-free ANN trained to satisfy the differential operator, initial conditions, and boundary conditions. A mesh free PDE simulation is extremely beneficial as obtaining solutions over fine meshes becomes infeasible as the dimension of the problem increases. In [12] ANNs are also used for solving high dimensional parabolic PDEs by casting the original PDEs as backwards stochastic differential equations and using an ANN as an approximation to the gradient of solutions.

Work has also been done directly utilizing the physics of fluid flow. For example, [13] illustrates utilizing physical principles of a system to inform deep learning models for sea surface temperatures. [14] provides an example of using constraints from a physical system in a loss function used to train an ANN. Our approach also relies on the physics of a system by having its computational roots in a physics-based numerical model; however, we extend physically constrained approaches and suggest a method for end-to-end training of an ANN while it is embedded in a numerical scheme.

Closely related to our work, [9, 15, 16] use neural-numerical hybrid approaches. In [15] spatial derivatives are estimated on low resolution grids using ANNs which are trained end-to-end. Sections 6 and 7 of [16] outline a neural architecture used to train parameters in a dissipative *Rusanov* flux scheme for the Euler equations with a relatively coarse spatial grid over early time steps. In [16] the entire numerical scheme is recast as a neural network by representing each computation in the Rusanov scheme with a neural network. In order to stay consistent during training, the neural architecture is held in place with only parameters corresponding to numerical viscosity coefficients allowed to change during training. The parameters are optimized using stochastic gradient descent (SGD) which is advantageous when compared with performing brute force sweeps over a mesh of 30 parameters (corresponding to the viscosity coefficients in the Rusanov scheme). Formulating the entire solver as a neural network allowed the required gradients to be calculated with backpropagation. The neural architecture in [16] is different from the approach we present in this work. We instead retain the majority of an existing hydrocode used with a traditional numerical method in its current state, and add a single neural component. We then use a differentiable programming (section 1.3) paradigm to compute gradients required to train the neural component. Moreover we aimed to extend beyond a parameter search with the goal of exploring wider ranges of functional forms of compensatory mechanisms within a numerical scheme using neural functions rather than using a neural framework to find optimal parameters associated with a particular scheme.

## 1.2. The need for online training of hybrid simulators

In this work, a ‘hybrid’ simulation refers to a numerical model in which one or more components of the underlying scheme are replaced with neural networks or other data-driven components. A primary motivation of hybrid simulation is to replace a component of a numerical model whose form may be poorly known with a data-driven component that can learn to improve upon existing numerical schemes. In some cases, it is possible to learn a data-driven component offline by training it to some higher-fidelity reference data before embedding the trained component into a hybrid code. For example, it may be possible to learn a fluid dynamics turbulence closure scheme by training an ANN to reproduce data from a high-resolution direct numerical simulation [17, 18], or to learn a sub-grid scale cloud parameterization scheme for an atmospheric climate model by training an ANN to reproduce the output of a high-resolution global cloud-resolving model [19–21]. Once trained to such pre-simulated offline reference data sets, these neural network emulators of higher-fidelity physics can then be inserted into coarser-resolution models to run online as a hybrid simulation, with the ANN component replacing existing numerical approximations to unresolved processes.

In other cases, however, there is no high-fidelity reference data for the behavior of AV, because AV is not derived from the physics of the system; it is an artificial term added to improve numerical aspects of the discrete solutions. Therefore, there is no ‘true’ viscosity data set against which to train the ANN. Instead, we learn the performance of a given AV scheme implicitly through its effect on the final solution generated by the hybrid model; although there is no reference ‘viscosity’ data from which to compute an AV directly, we can evaluate the predictions of a hybrid hydrocode to determine whether numerical artifacts such as spurious oscillations are being suppressed. In a hybrid model that replaces the AV numerical component with an ANN, this requires online training, by which we mean wrapping the whole hybrid simulator with an embedded ANN component in a training loop, in order to evaluate its performance and backpropagate its prediction errors to update the ANN weights. (We outline how this is done in section 3.)

The need for online training is far more general than the particular numerical setting examined in this paper, however. To further motivate the importance of our approach, we discuss other settings in which online training of a hybrid model may be necessary or advantageous. Many complex simulations contain

physical approximation schemes or parameterizations involving ad-hoc or purely empirical functions whose forms were chosen to invoke good simulation behavior, rather than being derived from any theoretical consideration. One example is in the K-profile parameterization (KPP) vertical mixing scheme of global ocean circulation models, which approximates unresolved turbulent mixing processes as a form of vertical diffusion [22]. KPP contains an arbitrary ‘shape function’ whose role is to artificially enhance or suppress vertical mixing at different ocean depths to compensate for other limitations in the mixing approximation. Although the shape function in KPP plays a more physical role than AV does in numerical hydrodynamics, there is still no reference data against which to train, because the shape function is an ad-hoc empirical construct.

In other cases, an offline reference data set for training may exist, but it is unclear how or whether it should be used within a hybrid code. Consider the example, previously discussed, of an ANN emulator of a high-resolution model embedded into a lower-resolution model as a sub-grid parameterization. An ANN that correctly emulates a high-resolution model may not produce the intended behavior when coupled to a second model, due to that model’s different structure and biases. This is related to the well-known phenomenon that parameter tuning of individual, standalone model components may fail when those components are coupled together [23, 24], and in a hybrid numerical-neural modeling context, has led to competing training and coupling approaches [20] vs. [19]. In such settings, it may be advantageous to supplement an offline pre-training stage with an online training stage to reduce biases or ‘coupling shock’ in the machine learning component once it is embedded interactively to a numerical solver, an approach that we take in this work.

To elaborate on the need for offline pre-training in combination with online training of hybrid models, we emphasize that the online training procedure requires forward simulations of the Euler equations with AV represented by a neural network. This presents numerical and training difficulties. Poor choices of AV will prevent a simulation from running to completion. For example, conservation laws can be violated resulting in the termination of a simulation. Hence, a randomly initialized ANN will most likely add undesirable values of viscosity in an AV scheme causing numerical instabilities, inconsistencies, and simulation failure. Additionally, online training is expensive since it requires forward simulations of the SGH scheme. Our solution, discussed in more detail in the methods, is to pre-train an ANN to reproduce the existing VonNeumann-Richtmeyer AV scheme currently used in our hydrodynamic simulations. This does not require expensive forward simulations of the full hydrocode, and initializes the ANN into a known, physically reasonable state. Furthermore, during the offline training the inputs to the ANN can be sampled in a space-filling manner [25, 26], rather than restricting the inputs to states generated by a physics model. Space-filling sampling allows for an ANN that has the potential to generalize to problems with conditions that may not be present (or hard to achieve) in states generated in an online training setting. Obtaining an ANN that is generalizable is important as hybrid modeling is expanded to wider ranges of test problems and applications.

### 1.3. Differentiable programming for online training of hybrid simulators

Our approach embeds a single machine learning component in an SGH numerical scheme which produces a hybrid model that is largely based on physics. An ANN, by itself, is typically trained through the backpropagation algorithm [27, 28], which is a form of parameter optimization based on gradient descent over a defined loss function. Modern machine learning frameworks such as TensorFlow [29] and PyTorch [30] compute the necessary gradients of the loss function with respect to ANN weights by constructing a computational graph of the neural network’s forward pass and traversing it backwards to incrementally compute gradients via the chain rule, known as reverse-mode automatic differentiation (AD) [31]. (Other styles of AD are possible besides graph construction, such as source-to-source transformation [32–34] or continuation-passing [35].)

In a hybrid physical-neural simulation model containing a machine learning component, the ANN weights effectively become new tunable parameters of the numerical solver. In order to train a hybrid model via gradient descent, it is necessary to differentiate a loss function, comparing the simulation output to training data, with respect to the neural network weights. This entails differentiating the entire hybrid code, not just the ANN, with respect to the ANN weights. If the code were entirely a numerical PDE solver, differentiating a functional of the PDE solution with respect to some of its parameters would correspond to the well-known construction of an adjoint PDE system.

Differentiating through our hybrid solver to train a neural network embedded in an existing numerical scheme is possible using differentiable programming [34], which allows for differentiation of arbitrary programs (as opposed to functions). This simply amounts to a recognition that AD is not limited to the particular ANN architectures to which backpropagation is traditionally applied, but rather can apply to general program structures, including the loops, conditional statements, and mathematical function calls

that comprise scientific numerical codes. Using the Flux and Tracker packages [36, 37] of the Julia scientific programming language, we apply reverse-mode AD to our entire hybrid simulator, constructing a computational graph through both the numerical PDE and ANN portions of the code. This corresponds to computing an adjoint through the PDE portion of the code, and backpropagating through the ANN portion of the code. With respect to the PDE portion, it corresponds to a ‘discretize-then-adjoint’ approach [38], in which an adjoint is constructed from an existing discretized PDE solver, as opposed to an ‘adjoint-then-discretize’ approach in which a continuum adjoint PDE would be constructed from the continuum Euler equations and then discretized [39].

#### 1.4. Outline

This remainder of the paper is organized as follows: In section 2 the inviscid Euler equations and associated shock tube-type Riemann problems are reviewed. In section 2 we also provide a brief description of the Lagrangian SGH scheme with AV that we will apply throughout the remainder of this work. Section 3 provides necessary background on ANNs and describes the offline and online training of the ANN. Numerical results are presented in section 4. In section 5 we describe several challenges associated with training an ANN in the context of hybrid numerical methods as well as future directions motivated by this work.

## 2. Inviscid fluid dynamics

The Euler equations of fluid dynamics [1, 2] can be formulated in a Lagrangian frame, in which case the movement of individual control volumes are tracked. In one-dimension, the Euler equations in the Lagrangian frame take the form

$$\begin{aligned} u &= \frac{dx}{dt} \\ \frac{d\rho}{dt} &= \rho \nabla \cdot u \\ \rho \frac{du}{dt} &= -\nabla p \\ \rho \frac{d\varepsilon}{dt} &= -p \nabla \cdot u. \end{aligned} \quad (1)$$

Here  $x$ ,  $u$ ,  $\rho$ ,  $p$ , and  $\varepsilon$  represent the position, velocity, mass density, pressure, and specific internal energy, respectively, of a *marked* fluid particle. The equations of motion (1) are also supplemented with a thermodynamic equation of state (EOS). Throughout we assume the EOS assumes the form of an ideal gas

$$p = \mathcal{P}(\rho, \varepsilon) \triangleq (\gamma - 1)\rho\varepsilon, \quad (2)$$

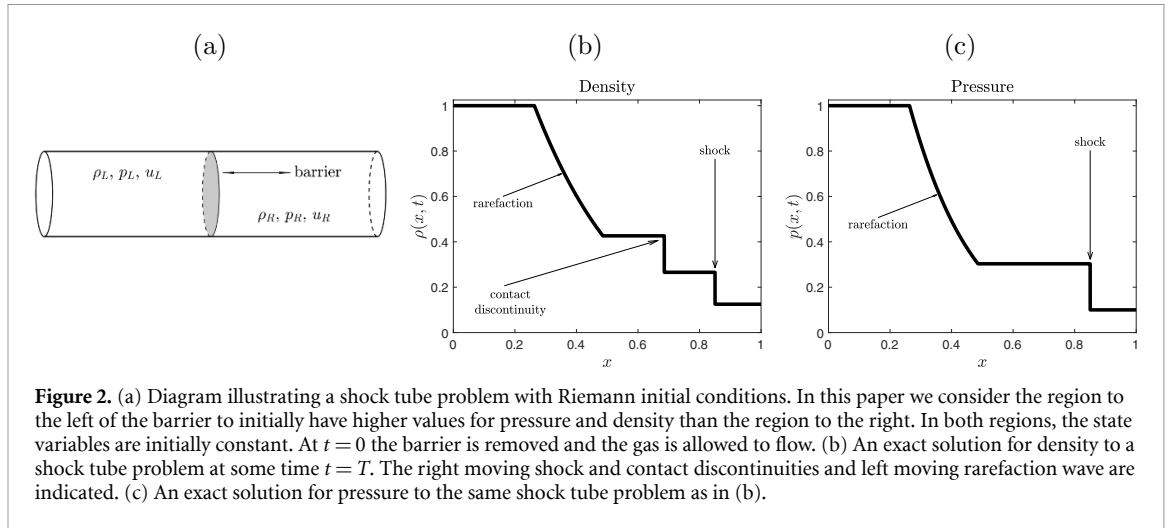
where  $\gamma$  is the ratio of specific heats at fixed volume and pressure.

### 2.1. The shock tube problem

The Euler equations can be used to model one-dimensional shocktube experiments [40]. Shocktube problems have exact solutions, which can be used as reference solutions to determine the accuracy of any numerical scheme. These simplified problems also exhibit many features of more complex flows including rarefaction, contact, and shock waves. For a detailed presentation of applying the method of characteristics to obtain exact solutions to shocktube problems, which belong to the more general class of Riemann problems for the Euler equations, see Toro [2].

Physically, a shock tube experiment involves a cylinder filled with two species of gas separated by a thin barrier or membrane. At time  $t = 0$  the barrier is removed and the gas is allowed to flow. Assuming that the gas species to the left of the barrier has higher pressure, then these initial conditions drive a *shock wave* that propagates into the lower pressure gas on the right. Across the shock wave the flow exhibits a jump discontinuity in all state variables. Behind the shock wave at the interface between the two gases, a *contact discontinuity* (that also propagates to the right) separates the two gas species. At the contact surface the density exhibits a jump discontinuity, but velocity and pressure remain continuous. In addition to the two right-moving discontinuities, there is a left moving *rarefaction wave* that forms a continuous transition connecting the left most high pressure state and the contact wave. Figure 2 illustrates the different features observed in the shock tube.

Numerical methods associated with solving the Euler equations and other hyperbolic PDEs that admit discontinuous solutions must be designed to handle numerical inconsistencies that arise as a result of the



breakdowns in continuity, e.g. shock waves and contact structures. In this work, we focus on resolving the solution near the chock in the setting of shocktube-type Riemann problems. In particular, the challenge is maintaining a sharp solution profile at the shock and while reducing the aforementioned spurious oscillations in the solution (see section 2.2). Numerical methods that satisfy both properties are called high resolution methods as described in [1].

It is possible to compute exact solutions to the shock tube problem using iterative schemes [2]. In this work we used the FORTRAN code provided at ([http://cococubed.asu.edu/code\\_pages/exact\\_riemann.shtml](http://cococubed.asu.edu/code_pages/exact_riemann.shtml)) to generate reference solutions. Hereafter, reference solutions are referred to as *exact* solutions. Exact solutions are a useful starting point for developing numerical methods since they provide a reference for verifying the performance of a new method. Moreover, for the work presented in this paper, the exact solution is invaluable as it is additionally used as reference data for training an ANN.

### 2.2. Staggered grid hydrodynamics

The SGH scheme with AV that applied is described in [3, 6, 41]. We provide a brief outline of the major steps below.

The discrete equations are solved on a pair of so-called staggered grids in which the spatial domain  $\Omega$  is divided into cells. The first grid divides  $\Omega$  into cells  $[x_i^n, x_{i+1}^n]$  with coordinates  $x_i^n$ . The second grid divides  $\Omega$  into cells  $[x_{i+1/2}^n, x_{i+3/2}^n]$  with coordinates given by the cell centers  $x_{i+1/2}^n = 0.5(x_i^n + x_{i+1}^n)$ . In the Lagrangian frame, since the mesh moves with the velocity field of the flow, the coordinates depend on both space and time indexed here by  $i$  and  $n$ , respectively. In the staggered-grid framework, velocities  $u_i^n$  are calculated at cell nodes  $x_i^n$  defined on the first grid, while density, pressure, and internal energy are calculated at cell centers  $x_{i+1/2}$  on the dual grid. Cell-centered quantities are correspondingly denoted  $\rho_{i+1/2}, p_{i+1/2}$ , and  $\varepsilon_{i+1/2}$ . These quantities represent volume-averages over each computational cell.

The solution is calculated in planar geometry using a second-order in time and space scheme outlined in [6, 41]. In the following equations, the subscripts denote the spatial discretization and the superscripts denote the temporal indexing. To solve for the state variables at  $t^{n+1} = t^n + \Delta t$ , first compute the predictor values:

$$u_i^{*,n+1} = u_i^n - \Delta t \frac{p_{i+1/2}^n - p_{i-1/2}^n}{m_i} \tag{3}$$

$$u_i^{*,n+1/2} = \frac{u_i^n + u_i^{*,n+1}}{2} \tag{4}$$

$$x_i^{*,n+1} = x_i^n + \Delta t u_i^{*,n+1/2} \tag{5}$$

$$\rho_{i+1/2}^{*,n+1} = \frac{m_{i+1/2}}{x_{i+1}^{*,n+1} - x_i^{*,n+1}} \tag{6}$$

$$\varepsilon_{i+1/2}^{*,n+1} = \varepsilon_{i+1/2}^n - p_{i+1/2}^n \Delta t \frac{u_{i+1}^{*,n+1/2} - u_i^{*,n+1/2}}{m_{i+1/2}} \tag{7}$$

$$p_{i+1/2}^{*,n+1} = \mathcal{P} \left( \rho_{i+1/2}^{*,n+1}, \varepsilon_{i+1/2}^{*,n+1} \right). \tag{8}$$

Then the predictor values are adjusted with a corrector stage, where the pressure is replaced by its predictor value above,  $p_{i+1/2}^n \mapsto p_{i+1/2}^{*,n+1}$ :

$$u_i^{n+1} = u_i^n - \Delta t \frac{p_{i+1/2}^{*,n+1} - p_{i-1/2}^{*,n+1}}{m_i} \tag{9}$$

$$u_i^{n+1/2} = \frac{u_i^n + u_i^{n+1}}{2} \tag{10}$$

$$x_i^{n+1} = x_i^n + \Delta t u_i^{n+1/2} \tag{11}$$

$$\rho_{i+1/2}^{n+1} = \frac{m_{i+1/2}}{x_{i+1}^{n+1} - x_i^{n+1}} \tag{12}$$

$$\varepsilon_{i+1/2}^{n+1} = \varepsilon_{i+1/2}^n - p_{i+1/2}^{*,n+1} \Delta t \frac{u_{i+1}^{n+1/2} - u_i^{n+1/2}}{m_{i+1/2}} \tag{13}$$

$$p_{i+1/2}^{n+1} = \mathcal{P} \left( \rho_{i+1/2}^{n+1}, \varepsilon_{i+1/2}^{n+1} \right). \tag{14}$$

It is important to note that in the Lagrangian frame, in contrast to the Eulerian frame, the mass of a cell  $m_{i+1/2}$  is constant (as is the mass at the cell edges  $m_i = (m_{i-1/2} + m_{i+1/2})/2$ ), so there is no need for the temporal superscript on the mass variable. Additionally one must specify boundary conditions; in this work we fix the velocity at the boundary to be zero,  $u_1 = u_{n+1} = 0$ . The time step  $\Delta t$  is chosen to satisfy a standard CFL condition with safety coefficient  $\lambda$ .

Figures 2(b) and (c) show the exact solutions for density and pressure in the Sod shock tube problem [42], a Riemann problem with discontinuous initial data:

$$(\rho_0, p_0, u_0) = \begin{cases} (1.0, 1.0, 0.0), & x < 0.5 \\ (0.125, 0.1, 0.0), & x > 0.5 \end{cases}, \tag{15}$$

solved over the spatial domain  $\Omega = [0, 1]$  with the initial barrier at  $x = 0.5$ . The corresponding numerical solutions shown in figure 1 and those used in the remainder of this paper are computed with  $N = 100$  cells. The initial conditions for the Sod test problem result in a relatively mild shock, which is useful for developing SGH numerical methods thus we consider the Sod problem for the rest of this paper and focus on reducing pre-shock oscillations (figure 1(a)) while maintaining a sharp shock resolution without excessive smearing. Since the velocity variable exhibits the largest, most rapid, oscillations the exact solution for velocity was used as a reference solution during the online training of the ANN.

### 2.2.1. Artificial viscosity

In the presence of shockwaves, the equations (3)–(14) are augmented with AV [3, 4, 6, 43] to reduce the formation of spurious oscillations behind the shock. The AV term, denoted by  $q$  is added to the pressure,  $p \mapsto p + q$ , in equation (1), and it acts as a dissipative mechanism. In this work we apply a standard quadratic-linear form of the AV with the formulation

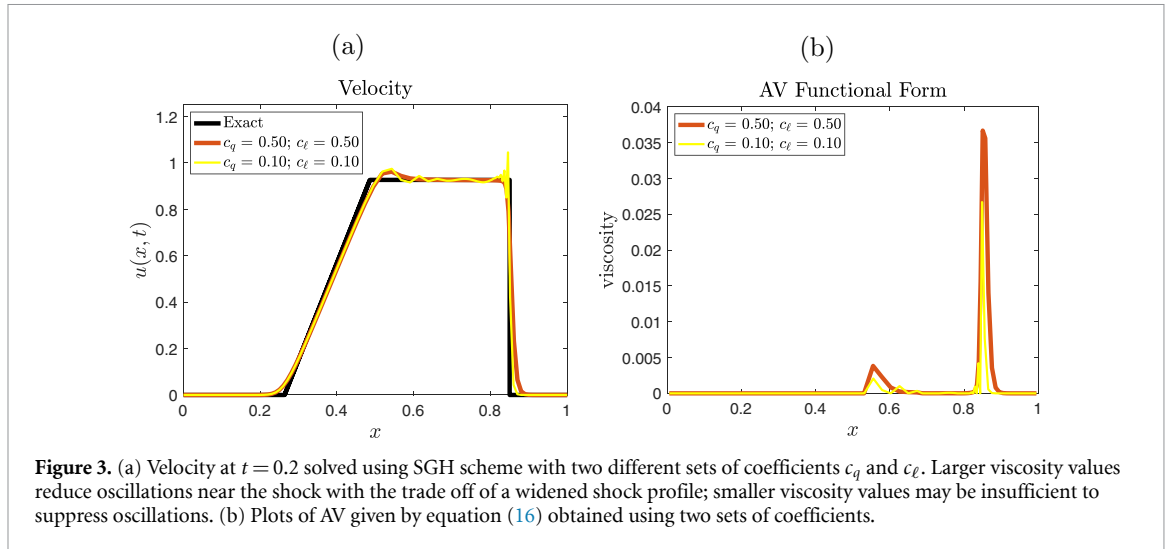
$$q_{i+1/2}^n = \begin{cases} (c_q |\Delta u_i^n| + c_s c_\ell) \rho_{i+1/2}^n |\Delta u_i^n|, & \Delta u_i^n \leq 0 \\ 0, & \Delta u_i^n > 0 \end{cases} \tag{16}$$

as suggested in [43]. The inputs into the  $q(\cdot)$  function are  $(\Delta u, \rho, c_s)$  where  $c_s$  represents the sound speed:

$$c_s = \sqrt{\gamma \frac{p}{\rho}}. \tag{17}$$

In the discrete equations (3)–(14) AV is a local quantity defined at cell centers,  $q_{i+1/2}$  used to augment pressure  $p_{i+1/2} \mapsto p_{i+1/2} + q_{i+1/2}$ . The inputs into  $q(\cdot)$  given in (16) are therefore also the discretized





**Figure 3.** (a) Velocity at  $t = 0.2$  solved using SGH scheme with two different sets of coefficients  $c_q$  and  $c_\ell$ . Larger viscosity values reduce oscillations near the shock with the trade off of a widened shock profile; smaller viscosity values may be insufficient to suppress oscillations. (b) Plots of AV given by equation (16) obtained using two sets of coefficients.

quantities  $(\Delta u_i^n; \rho_{i+1/2}^n, c_{s,i+1/2}^n)$  where  $\Delta u_i^n$  is the difference in velocity between cell endpoints,  $\Delta u_i^n = u_{i+1}^n - u_i^n$ .

The quadratic coefficient  $c_q$  and the linear coefficient  $c_\ell$  are parameters chosen by the user. Since these coefficients are introduced as part of the numerical scheme, they are not constrained by the physical properties of the gas being modeled. Yet, naive choices can lead to overly oscillatory or overly smeared solutions as well as other non-physical characteristics in the solution. Figure 3(a) shows numerical solutions for velocity with AV added using two sets of coefficients. As illustrated in figure 3(a) choosing values of  $c_q$  and  $c_\ell$  larger in magnitude tends to produce numerical solutions with reduced oscillations near the shock at the cost of a smeared shock profile. In contrast, the profile of solutions obtained using coefficients relatively smaller in magnitude can fail to sufficiently reduce oscillations. Figure 3(b) shows the functional form of AV using equation (16) and it can be seen that less AV is added at the shock when coefficients  $c_q$  and  $c_\ell$  are smaller in magnitude.

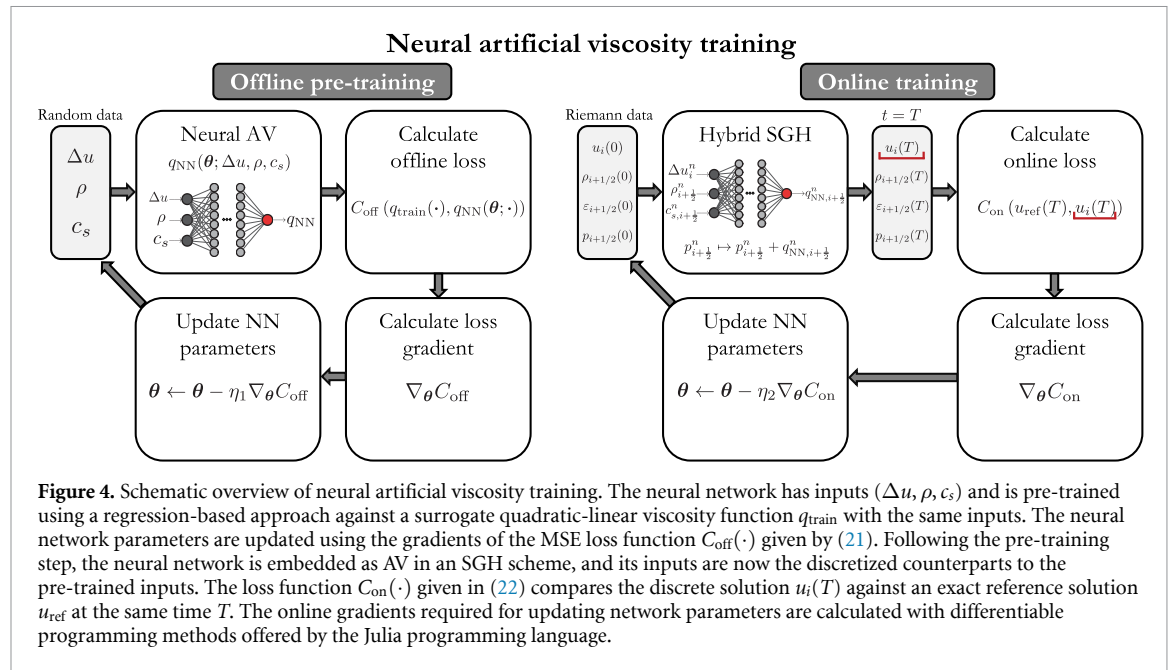
Figure 3 illustrates that there is a trade off between oscillation suppression and shock maintenance when choosing the coefficients  $c_q$  and  $c_\ell$ . That is, adding excessive AV overly smears solutions, but adding insufficient AV fails to suppress oscillations [6]. Since the choices for the coefficients  $c_q$  and  $c_\ell$  are not physically constrained, one may choose to optimize these parameters so that they minimize the aforementioned trade-off between overly oscillatory and overly smeared solutions. This can be achieved with a brute force parameter sweep of the coefficients. However, this derivative-free approximation approach becomes infeasible in higher dimensions. Moreover, the optimal trade off is constrained by the chosen functional form of the  $q(\cdot)$  function given by (16). Therefore, the goal of this work is to replace the linear-quadratic AV function with an ANN trained to minimize a loss function designed to capture the trade-off between highly oscillatory and overly smeared solutions. Our hypothesis is that an ANN allows for more functional flexibility than the user-specified  $q(\cdot)$  function with constant coefficients.

### 2.2.2. Generalizing artificial viscosity

As mentioned in section 2.2.1 the optimal of balance pre-shock oscillations with a sharp shock profile is constrained by the choice of viscosity function. Therefore, our aim was to explore a more general functional form for AV by replacing an explicit formula with a learned neural viscosity function represented by an ANN.

In the context of ANNs, the universal approximation theorem [44] states that a feedforward neural network with sigmoidal activation can approximate any continuous function up to an arbitrary level of precision, provided there are enough nodes in the network. Given this general theorem, the idea behind our approach is that the ANN will approximate a functional form of the AV that can better optimize a prescribed loss function to determine an optimal AV value  $q$ . In particular, by replacing a fixed functional form of AV with a more general, data-driven ANN, we allow for an exploration to more general functional forms for AV.

As a starting point for the ANN, we constructed a network to have the same inputs as the  $q(\cdot)$  function in (16), namely  $\Delta u$ ,  $c_s$ , and  $\rho$ . The ANN then produces a single scalar output corresponding to the AV term that is used to augment pressure in (3)–(14). That is, if we let  $q_{\text{NN}}(\cdot)$  represent the ANN that will serve as AV,



we have that  $q_{NN} : \mathbb{R}^3 \rightarrow \mathbb{R}$ . In particular, note that  $q_{NN}$ , just is a potentially non-linear function with a neural representation. In close analogue to (16) we similarly formulate the neural viscosity as a local quantity:

$$q_{NN,i+1/2}^n = F(\theta; \Delta u_i^n, \rho_{i+1/2}^n, c_{s,i+1/2}^n). \quad (18)$$

The function  $F(\theta; \cdot)$  represents a learnable neural function whose construction is described in more detail in section 3.

### 3. Artificial neural networks

ANNs are functions  $F(\theta; \mathbf{x})$  that map input variables  $\mathbf{x}$  to output variables  $\mathbf{y}$ . In this work we used multi-layer perceptrons, networks that are densely connected and for which each layer's output successively serves as the next layer's input given by the relation

$$\mathbf{v}^{(\ell+1)} = \sigma(W^{(\ell)}\mathbf{v}^{(\ell)} + \mathbf{b}^{(\ell)}). \quad (19)$$

Here  $\mathbf{v}^{(\ell)}$  represents the output of the  $\ell$ th layer in the network.  $W^{(\ell)}$  is the weight matrix where  $W^{(\ell)}[i, j]$  represents the connection from the  $j$ th computational node in layer  $\ell$  to the  $i$ th node in layer  $\ell + 1$ . The components in the vector  $\mathbf{b}^{(\ell)}$  represents the bias that shifts the threshold for the activation function  $\sigma(\cdot)$  for each node in layer  $\ell + 1$ . The collections  $\{W_{\ell}\}$  and  $\{\mathbf{b}_{\ell}\}$  together are represented concisely by  $\theta$ .

ANNs are trained by optimizing the parameters  $\theta$  to minimize a loss function  $C(\theta; \mathbf{x}, \mathbf{y})$  using a collection of training data  $\{\mathbf{x}, \mathbf{y}\}$ . The cost function is typically minimized through variants of SGD in which partial derivatives of  $C(\theta, \mathbf{x}, \mathbf{y})$  are calculated with respect to each parameter in  $\theta$ . ANNs have been particularly successful in regression and classification settings. For an introduction to neural networks and other machine learning topics, the book by Strang [45] and the references within are a good resource. The computing software Julia and its Flux package [36] offers first class support for training ANNs and calculates derivatives using AD [31]. The Flux package in Julia was used to train the ANNs in this work. The schematic overview for the offline and online training of the ANN is shown in figure 4. In the next two sections, we provide more details for each training step.

#### 3.1. Offline pre-training

The offline training was done for two primary reasons. The first is to understand what is required, particularly in terms of the neural architecture (number of hidden layers, number of nodes, and activation functions), for an ANN to reproduce the traditional quadratic-linear AC form, and to test whether small errors made by the ANN approximation to the existing AV scheme will drastically worsen results. For example, naive choices for AV can lead to nearly immediate failure of the SGH scheme to run to completion—it was essential to have a starting point for AV to be confident that the hydrocode could reliably reproduce solutions before attempting to improve AV. The second motivation is that the quadratic-linear

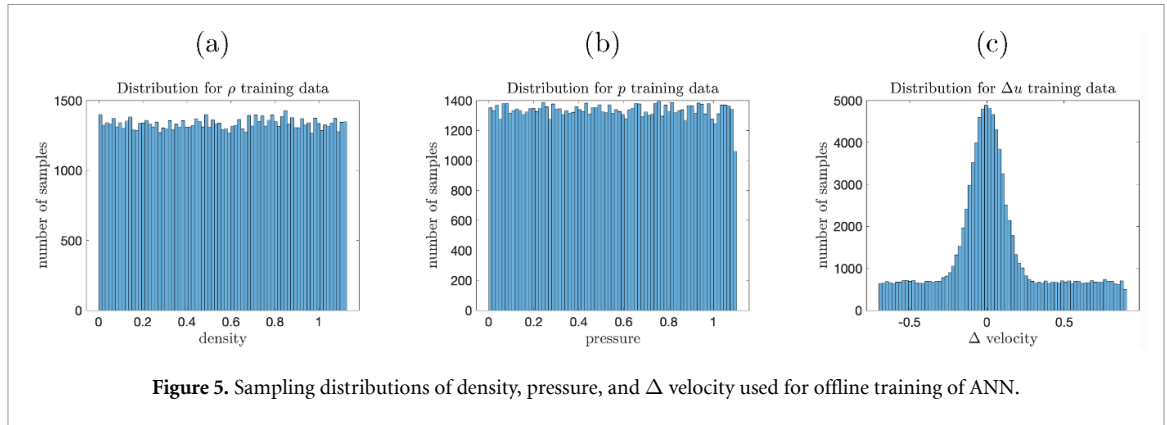


Figure 5. Sampling distributions of density, pressure, and  $\Delta u$  velocity used for offline training of ANN.

choice for AV has been studied extensively and has been shown to produce reasonably non-oscillatory solutions and with minor modifications can be used in more than one dimension [43].

The ANN was trained in a classic regression setting against the surrogate  $q(\cdot)$  function given by (16) with parameters  $c_q = 0.3$  and  $c_\ell = 0.3$ . These parameters were chosen by previous work done in [7] to minimize  $L_1$ -error for the density state variable. Additionally, this choice of coefficients corresponded to near-minimum  $L_1$  error for the remaining state variables with reasonably suppressed pre-shock oscillations. The offline pre-training was done purely against the surrogate  $q(\cdot)$  function, independent of the SGH scheme, and hence is termed **offline** training.

The surrogate quadratic-linear  $q(\cdot)$  function (16) has inputs  $(\rho, \Delta u, c_s)$  and maps to a single scalar output, hence the ANN was structured the same way—to have three inputs whose layers terminated to a single output node. To generate training data in the offline setting we first had to generate input variables  $(\rho, \Delta u, c_s)$ . First values for  $\rho$  (density) were sampled uniformly from the interval  $[0, 1.125]$ . These values extend to values of density that are not attained in the Sod shock tube problem, but ensured that ANN would not be forced to extrapolate for density values not present in offline training when used in the hydrocode. Next values for  $p$  (pressure) were sampled uniformly from the interval  $[0, 1.1]$ . While pressure is not a direct input into the quadratic-linear  $q(\cdot)$  function, the values for  $\rho$  and  $p$  were used together to generate values for sound speed,  $c_s$ , using equation (17) for the speed of sound.

Stratified sampling with two sources was used to generate samples for  $\Delta u$ . The first source was a uniform sample from the interval  $[-0.9, 0.7]$  and represented an extension of a plausible range of values  $\Delta u$  could achieve in the Sod problem. The second source for the  $\Delta u$  sample came from sampling from a normal distribution with mean 0 and standard deviation 0.01. The normal distribution closely matched the distribution of values for  $\Delta u$  obtained from a numerical solution to the Sod shock tube problem with AV given by (16). Sampling  $\Delta u$  partially from the normal distribution was especially crucial given that in the solution to the Sod shock tube problem  $\Delta u$  takes on the value zero at a large number of computational cells and care was required to avoid under sampling at values near zero. The uniform sampling portion of  $\Delta u$  helped maintain the ANN's ability to generalize when  $\Delta u$  is not zero (specifically at the shock). Sampling distributions for density, pressure, and change in velocity are shown in figure 5. Values for training outputs were obtained by evaluating the surrogate quadratic-linear  $q(\cdot)$  function using the randomly generated training data,  $(\rho, \Delta u, c_s)$ , as inputs into (16). The space-filling sampling described above improves the likelihood that the offline-trained ANN can generalize to problems with different density ratios and shock intensities (such as the Le Blanc test problem [46]).

Overall  $N = 100\,000$  data points were generated and split with  $N_{\text{train}} = 80\,000$  points used for training and  $N_{\text{test}} = 20\,000$  points used for validation. In accord with standard ANN training practices values for each input variable  $\rho$ ,  $\Delta u$ , and  $c_s$  were standardized to have mean 0 and standard deviation 1. The training outputs were transformed by taking the 5th root of the output values,  $\sqrt[5]{q(\cdot)}$ . Other transformations (standardization, log-transformations, and other values for  $n$ th roots) were tested for the output data. Ultimately the 5th root was chosen because this transformation exaggerated differences between zero and numerical values for  $q$  very near zero, but the transformation allowed for a flexibility in the ANN that would be utilized during the online training of the network.

The ANN was initialized with bias vectors  $\mathbf{b}^{(\ell)} \equiv 0$  and weight matrices  $W^{(\ell)}$  uniformly distributed with mean zero and small variance. The network had a depth of four hidden layers with each hidden layer having a width of 64. The activation function for each hidden layer node was the leaky rectified linear unit function

$$\text{leakyReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0.01x, & x < 0 \end{cases}. \quad (20)$$

The ANN was trained using the *Adam* optimization method, a gradient-based method for optimizing stochastic objective functions [47]. Additionally mini-batch training with batch size  $|\mathcal{B}| = 32$  was used during the offline training. In the mini-batch process a sample (rather than a single point as in classic SGD) of size  $|\mathcal{B}|$  is taken from the training data and used to calculate gradients. During the offline training mean squared error (MSE) was used as a cost function:

$$C_{\text{off}}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^m (\hat{q}_i - q_i)^2 \quad (21)$$

where  $\hat{q}$  represents the output of the ANN. The training process was concluded once the MSE calculated on the validation set was within  $1.0 \times 10^{-5}$ . The entire offline training procedure (data generation, model training, and model saving) took between 1 and 2 min on a personal desktop (3.4 GHz processor and 16 GB RAM), yielding a neural AV function that serves as an initial viscosity model for the online training procedure.

### 3.2. Online training via differentiable programming

After an ANN that could reliably be used in the SGH scheme (see section 4) was obtained from offline training, the next step was to replace the surrogate  $q(\cdot)$  function with the offline pre-trained ANN in the hydrocode. The parameters in the ANN were updated by computing a loss function using an exact solution for velocity as a reference solution. This training is done while implementing the hydrocode and is hence called **online** training. It is important to note again that it is a significant advantage to have exact data (which is possible with the Euler equations) for the online training.

The loss function used in the online training was a linear combination of  $L_1$ -error and change in total variation  $\Delta TV$ :

$$C_{\text{on}}(\boldsymbol{\theta}) = \lambda_1 \Delta TV_u + \lambda_2 \|u\|_{L_1}, \quad (22)$$

where

$$\|u\|_{L_1} = \frac{\sum_i |u_{\text{ref},i} - u_{\text{NN},i}| \Delta x_i}{\sum_i |u_{\text{ref},i}| \Delta x_i} \quad (23)$$

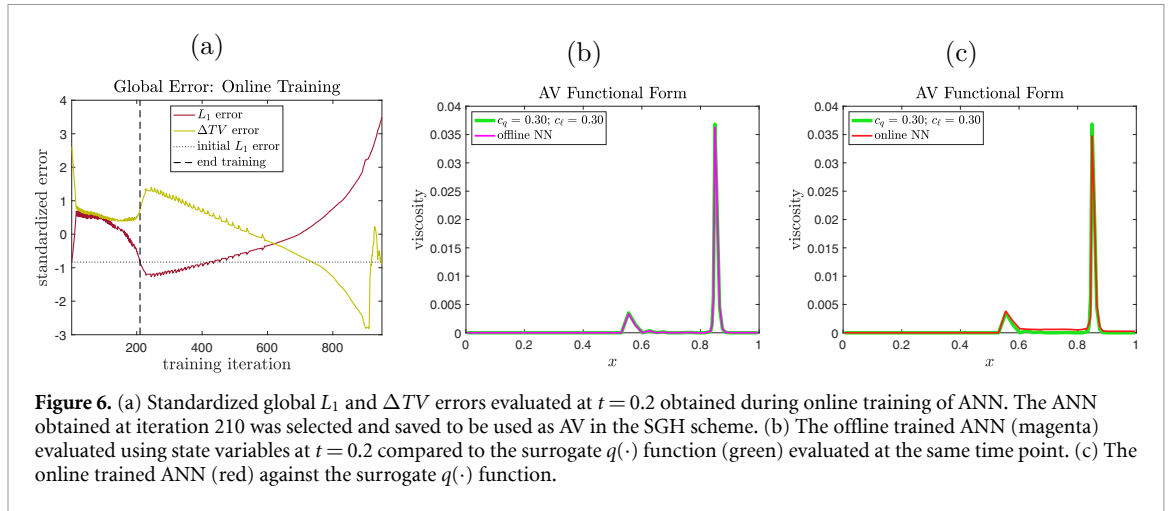
$$TV_u = \sum_i |u_{i+1} - u_i| \quad (24)$$

$$\Delta TV_u = \frac{|TV_{u_{\text{ref}}} - TV_{u_{\text{NN}}}|}{TV_{u_{\text{ref}}}}. \quad (25)$$

Here  $u_{\text{ref}}$  and  $u_{\text{NN}}$  represent the exact solution for velocity and the numerical solution for velocity using the ANN as viscosity. Lower  $L_1$  errors correspond to solutions with a sharp shock profile, while lower  $\Delta TV$  errors correspond to solutions with reduced oscillations, thus the loss function in (22) reflects the tradeoff between oscillation reduction and shock width maintenance. This loss function was used in [41] as an error function in an SGH with AV problem close to that discussed in this paper.

Since this is a hybrid numerical-neural model, taking the gradient of the loss function with respect to the ANN parameters requires differentiating through the whole hybrid model. In the physics model the differentiation is the *adjoint* and in the neural network it is *backpropagation*, but in both cases differentiating through the entire code is required to get the derivatives needed for gradient descent optimization, i.e. differentiable programming. We use reverse-mode tracing AD in the Flux package [37] in the Julia programming language [48] to facilitate the differentiation. Julia, a high-level, high-performance scientific computing language is ideal for writing codes that combine numerical simulations, such as differential equations solvers, with machine learning [34, 49, 50].

We emphasize that the online training procedure is quite different from the classical regression setting since we are no longer specifying a functional form for AV. Instead we are optimizing the ANN to minimize a loss function designed to take on large values for overly oscillatory solutions ( $\Delta TV$ -error) and overly smeared shock profiles ( $L_1$ -error). Specifically, the SGH scheme with the ANN as viscosity was ran forward until  $t = 0.05$  (approximately 50 time steps determined by CFL condition). The loss function (22) was calculated with  $\lambda_1 = 0.5091$  and  $\lambda_2 = 0.4909$  using the numerical and exact solutions for velocity at this time point. In general these coefficients introduce additional parameters that may need to be adjusted to a particular problem. The cost function was calculated locally at computational cells around the shock. Recall



**Figure 6.** (a) Standardized global  $L_1$  and  $\Delta TV$  errors evaluated at  $t = 0.2$  obtained during online training of ANN. The ANN obtained at iteration 210 was selected and saved to be used as AV in the SGH scheme. (b) The offline trained ANN (magenta) evaluated using state variables at  $t = 0.2$  compared to the surrogate  $q(\cdot)$  function (green) evaluated at the same time point. (c) The online trained ANN (red) against the surrogate  $q(\cdot)$  function.

that in the Lagrangian frame, the grid cell locations evolve in time (specifically accumulating in areas before the shock). Hence a shock-width of 0.05 was specified before training and the cost function was calculated at all cells that lied within this pre-specified distance of the shock. The parameters in the ANN were updated using AD (still using the capabilities of the Flux package in Julia). We stress that at no point in the online training is the surrogate  $q(\cdot)$  function explicitly used.

Since the loss function was computed locally at early time points in the solution, care was required to terminate training before the ANN would become overly specialized at the local grid cells and early stopping time. The goal of this work was to illustrate that modifying the functional form of AV can improve aspects of the numerical solution; optimizing the online training procedure was not a primary goal. Therefore global  $L_1$  and  $\Delta TV$  errors evaluated at a final stopping time of  $t = 0.2$  were used to determine the termination of the online training procedure; after each training iteration the system was allowed to evolve to the final time  $t = 0.2$  with the updated ANN and global  $L_1$  and  $\Delta TV$  errors were calculated and recorded at these points. After many training iterations these values were evaluated to decide a proper termination point for the online training. Figure 6(a) shows the tracking of global errors at the stopping time of  $t = 0.2$  during the online training. The online procedure for the Sod problem up to training termination lasted on the order of 30 s (personal desktop 3.4 GHz processor and 16 GB RAM). For problems with more intense shocks, the online training time would likely increase due to grid-refinement and time-stepping changes.

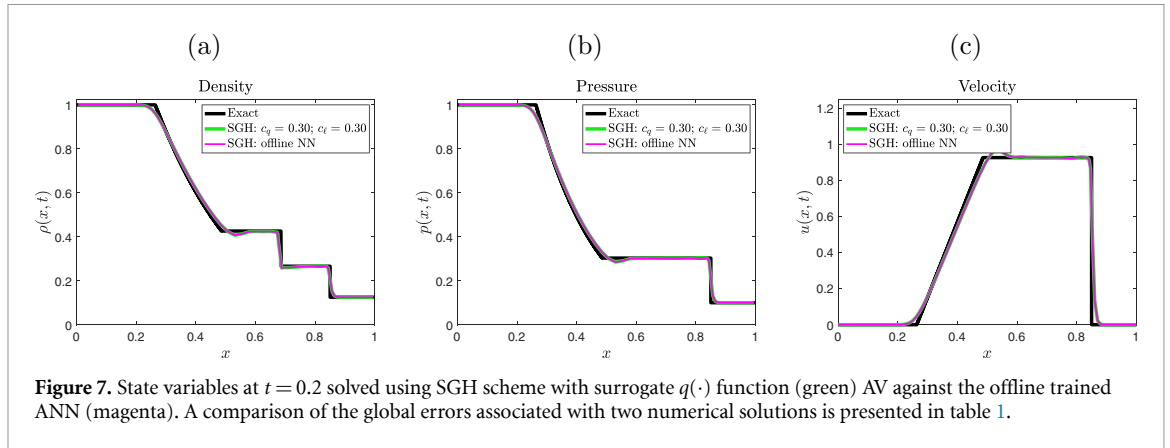
## 4. Results

### 4.1. An offline trained ANN emulates surrogate $q$ in Sod shock tube

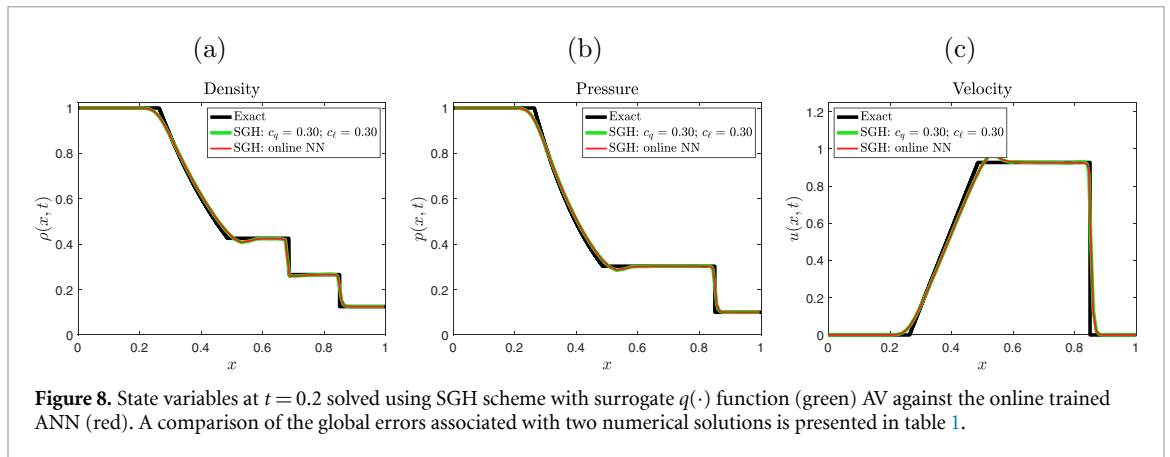
The first result of this work was successfully replacing the surrogate quadratic-linear  $q(\cdot)$  function with an offline trained ANN to act as AV in the SGH scheme. This was a critical step toward illustrating that the ANN could reliably be used as AV in the SGH scheme and produce empirically stable solutions. It was a possibility that errors from using an ANN to approximate AV could propagate leading to unstable solutions. Figure 7 shows solutions obtained using the quadratic-linear viscosity function given by (16) and solutions obtained using the offline trained ANN as AV. Observe that in these figures the two numerical solutions are closely aligned indicating that an ANN is able to replicate solutions obtained using the traditional quadratic-linear viscosity function. Errors computed against the exact solution are shown in table 1. The percent change in errors obtained with neural viscosity compared to errors obtained using quadratic-linear viscosity are small. The  $L_1$  error in the three state variables differ by less than 0.5%, and the  $\Delta TV$  errors differ by at most 1.15%. This indicated that it was possible to use a neural viscosity function in a numerical scheme, which was essential for the ultimate goal of performing end-to-end training of an ANN embedded in an SGH scheme. Figure 6(b) compares the functional forms of the quadratic-linear  $q(\cdot)$  function and the neural viscosity evaluated with the final state variables as inputs. The two plots are closely aligned again indicating that the offline pre-training produced an ANN that can mimic the behavior of the quadratic-linear viscosity function, and that errors presented by an ANN would not cause early termination of the SGH scheme.

### 4.2. An online trained ANN reduces pre-shock oscillations in Sod shock tube

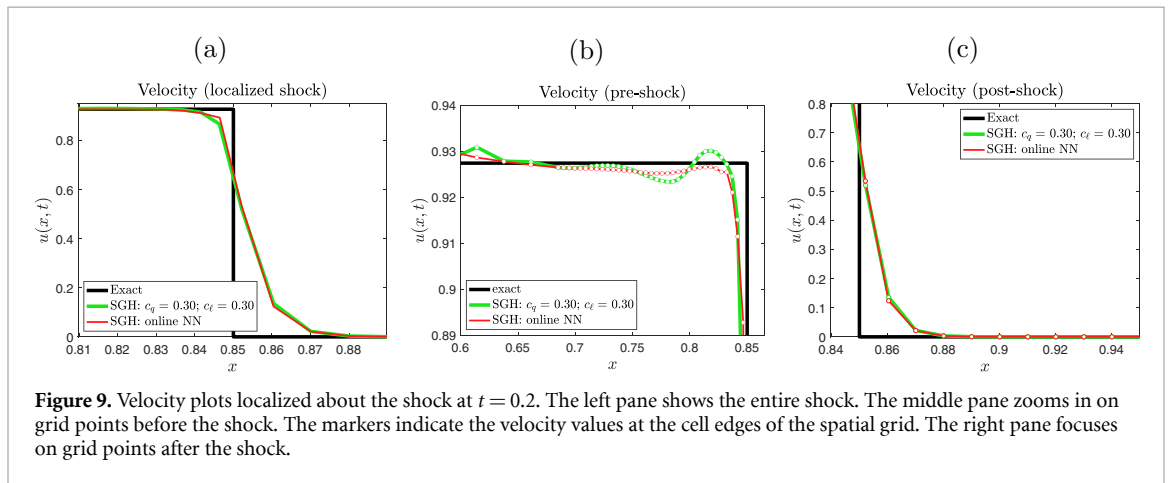
The purpose of the online training procedure was to direct an ANN to suppress pre-shock oscillations while maintaining a sharp shock profile. An implicit goal is maintaining precision across all phases of the solution, not just improving aspects at the shock. It is important for the ANN to be flexible enough to yield reliable



**Figure 7.** State variables at  $t = 0.2$  solved using SGH scheme with surrogate  $q(\cdot)$  function (green) AV against the offline trained ANN (magenta). A comparison of the global errors associated with two numerical solutions is presented in table 1.



**Figure 8.** State variables at  $t = 0.2$  solved using SGH scheme with surrogate  $q(\cdot)$  function (green) AV against the online trained ANN (red). A comparison of the global errors associated with two numerical solutions is presented in table 1.



**Figure 9.** Velocity plots localized about the shock at  $t = 0.2$ . The left pane shows the entire shock. The middle pane zooms in on grid points before the shock. The markers indicate the velocity values at the cell edges of the spatial grid. The right pane focuses on grid points after the shock.

results at regions away from the shock. Figure 8 illustrates that the online ANN maintains the global precision observed with the surrogate  $q(\cdot)$  function.

Figure 9 focuses on the behavior of the solution localized about the shock illustrating the suppression of pre-shock oscillations. Figure 9(a) highlights the preservation (and slight improvement) of the shock profile. Before the shock, the solution obtained using the ANN as AV,  $u_{NN}$  (red), lies above (and hence closer to the exact curve) the solution obtained using the surrogate  $q(\cdot)$  function,  $u_{QL}$ . After the shock the two curves remain closely aligned. Figure 9(b) emphasizes the reduction in pre-shock oscillations for  $u_{NN}$  compared to  $u_{QL}$ ; the solution  $u_{NN}$  is nearly monotone across this region.

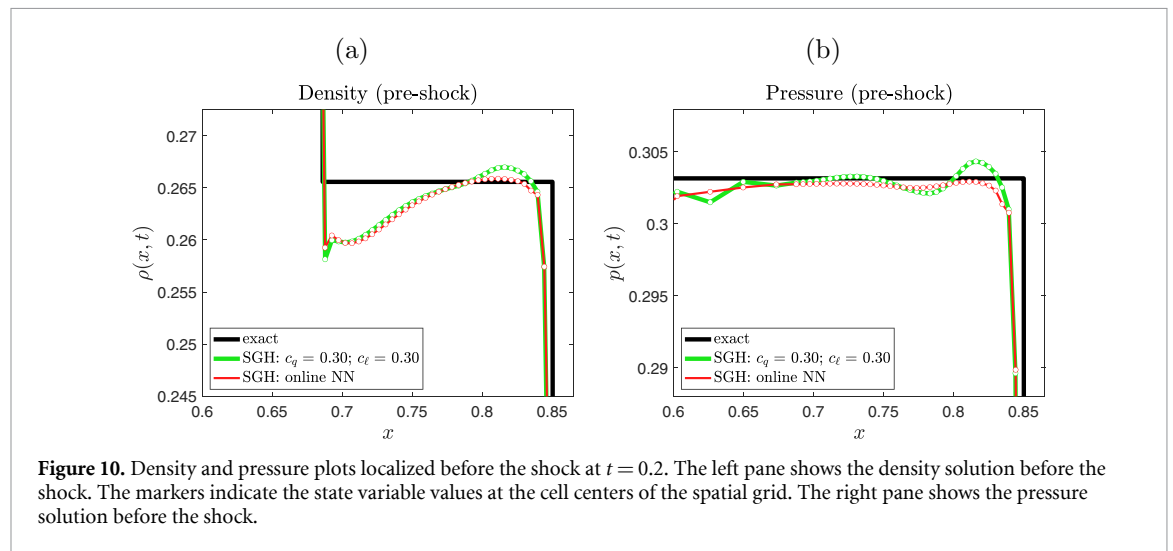
Despite being trained against an exact solution for velocity, oscillations were also suppressed in the other state variables. Figures 10(a) and (b) show the suppression of oscillations in density and pressure respectively by plotting the state variables localized at spatial cells before the shock. The reduction in oscillations for all state variables is quantified using the change in total variation,  $\Delta TV$  given by (25); the maintenance (and in fact slight improvement) of the shock profile is quantified using the  $L_1$  norm given by (23). Table 1 contains

**Table 1.** Global error values for different types of AV using error formulas given in (23)–(25). Percent changes are calculated against errors given by solutions obtained using the surrogate  $q(\cdot)$  function with parameters  $c_q = 0.3, c_\ell = 0.3$  labeled in the table by  $\star$ . The percent changes are calculated according to equation (26). Arrows indicate an increase (up arrow) or decrease (down arrow) in calculated error. The largest percent reduction in error for each metric is shown in bold.

AV	$\ \rho\ _{L_1}$		$\ p\ _{L_1}$		$\ u\ _{L_1}$	
	Error	% change from $\star$	Error	% change from $\star$	Error	% change from $\star$
$\star c_q = 0.3, c_\ell = 0.3$	0.0107	NA	0.0116	NA	0.0282	NA
Offline ANN	0.0107	↑ 0.02%	0.0116	↓ 0.10%	0.0283	↑ 0.33%
Online ANN	0.0105	↓ <b>1.18%</b>	0.0114	↓ <b>1.56%</b>	0.0275	↓ <b>2.50%</b>

AV	$\Delta TV_\rho$		$\Delta TV_p$		$\Delta TV_u$	
	Error	% change from $\star$	Error	% change from $\star$	Error	% change from $\star$
$\star c_q = 0.3, c_\ell = 0.3$	0.0619	NA	0.0479	NA	0.0531	NA
Offline ANN	0.0621	↑ 0.17%	0.0473	↓ 1.15%	0.0525	↓ 1.03%
Online ANN	0.0524	↓ <b>15.39%</b>	0.0377	↓ <b>21.29%</b>	0.0396	↓ <b>25.41%</b>



these values which were computed at  $t = 0.2$  against an exact solution. In order to measure the improvement of the solutions obtained using online trained neural viscosity compared to the solutions obtained with quadratic-linear function, the percent change was calculated according to the formula

$$\%change = \frac{error_{QL} - error_{NN}}{error_{QL}} \times 100. \tag{26}$$

Equation (26) was used for computing changes in each error measurement,  $L_1$  and  $\Delta TV$ , for each of the state variables and these values are shown in table 1.

### 5. Summary and discussion

Many numerical methods used in fluid dynamics will produce spurious oscillations near discontinuities and hence require additional modification, such as AV, to handle these special features. The additional terms and modifications have the advantage of allowing very general numerical methods to be specialized to particular problems and applications. However, in many cases the theoretical justifications for fixing the introduced control parameters to particular values is limited. Therefore, in this work we took a hybrid simulation approach and replaced a component of a numerical model, containing arbitrary control parameters, with a data-driven component that can learn to improve its performance with the proper training data.

In the particular context of Lagrangian SGH schemes for modeling flows with shock waves, AV introduces several options that must be supplied by a user. First, a functional form for AV must be specified. A given functional form also introduces control parameters that are not constrained by material properties of the gas or fluid. Therefore, in practice AV is tuned according to empirical success in improving model performance. However, the performance of the numerical scheme is immediately constrained by the

functional choice for AV. Using an ANN as a guide, we are able to explore a broader space of functions, potentially different from fixed AV formulations (linear-quadratic, for example)

Section 4 provides a preliminary proof-of-concept that a novel online training procedure of an ANN leads to a functional form of AV different from what is explicitly given in (16). Since there is no ‘true’ viscosity data set against which to train the ANN, we trained the ANN through the performance of a given AV scheme implicitly through its effect on the final solution generated by the hybrid model. To calculate gradients of the loss function with respect to parameters in the ANN we applied reverse-mode AD to the entire hybrid simulator using the Flux and Tracker packages of the Julia scientific programming language. We found that online training over early time steps of a simulation produced a neural AV function capable of reducing numerical oscillations in long-term hydrodynamic shock simulations.

As mentioned in section 2.2.1, adding larger amounts of viscosity tends to lead to solutions that are smeared across a wider range of computational cells. Figures 6(b) and (c) offer an interesting counter to this rule. The online trained ANN (figure 6(c)) adds slightly higher levels of AV across the spatial region containing the contact discontinuity before the shock, and adds *less* viscosity at the shock. This is interesting since in most cases adding less viscosity would lead to more oscillatory solutions. In fact, slightly altering the functional form of AV across the contact discontinuity helps to decrease oscillations behind the shock reducing the need for larger values of AV at the shock, helping to maintain a sharp shock profile. This evidence suggest that the online training of a hybrid model can aid in the exploration for alternative functional forms for viscosity that can improve the trade off between pre-shock oscillations and smearing of the shock profile.

When establishing the limitations of novel numerical methods, one is typically concerned with demonstrating various consistency, stability, and convergence properties. While these properties can be rigorously analyzed using a number of existing tools, e.g. Taylor series, for more traditional closed form discretization procedures, the ANN introduces new challenges to assessing model performance. In future work, one approach that could be used to assess convergence is to test an ANN-AV scheme over a wider suite of test problems and computationally measure convergence as an empirical starting point. Another possible approach to assessing the convergence of a hybrid model is to analyze the functional form of the neural viscosity and search for a closed form function with the same behavior as the neural AV. Such an explicit function could then be used to assess convergence of the model using traditional techniques. Moreover, a closed formulation increases the possibility of a functional interpretation of the neural viscosity set in the context of the underlying physical system, which is typically not possible with a deep ANN.

The Sod shock tube is a standard test problem for numerical schemes in 1D planar geometry. In [6] Noh formulates the same quadratic-linear viscosity functional for schemes in cylindrical and spherical geometries. In these geometries the radially symmetric Noh problem [6] is a commonly used test problem. Since the Sod problem was specifically considered in this work, simulations in non-planar geometries were not calculated. Given the same formulation of the surrogate quadratic-linear viscosity in cylindrical and spherical geometries, however, future works should test that the same methodologies presented here can be applied to schemes for 1D problems in such geometries.

Generalizing the neural AV scheme to higher dimensions will most likely require a reconfiguration of the neural network input–output structure. A common approach to improving AV schemes in higher dimensions is to formulate AV as a tensor instead of a scalar [43]. Similarly, a neural AV function should be adjusted accordingly and thus the input–output structure of the network would need to be changed. For example, in higher dimensions, we expect that components would need to be added to the AV function to account for the direction of discontinuity propagation. In a neural network this would correspond to additional input into the network to provide directional knowledge, and the output of the network would likewise need to indicate which computational cells need to be augmented with AV.

The theory of computing numerical solutions to PDEs is a rich and widely studied field equipped with many techniques that can be applied to a wide array of problems. The suggestion in this work is not to replace existing methods with purely machine learning based methods. Instead, we suggest adding data driven components to a model, when appropriate (e.g. a neural viscosity function), yielding a hybrid model that remains primarily based on the physics of the underlying problem. In this work we were able to train an ANN capable of reducing pre-shock oscillations in a hydrodynamic shock simulation. We believe these numerical results offer an early proof-of-principle that online differentiable training of hybrid numerical schemes with novel neural network components can improve certain performance aspects alongside conventional discretization techniques used in numerical schemes.

### Data availability statement

The data that support the findings of this study are available upon reasonable request from the authors.



## Acknowledgments

This research was supported in part by an appointment with the National Science Foundation (NSF) Mathematical Sciences Graduate Internship (MSGI) Program sponsored by the NSF Division of Mathematical Sciences. This program is administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and NSF. ORISE is managed for DOE by ORAU. All opinions expressed in this paper are the authors' and do not necessarily reflect the policies and views of NSF, ORAU/ORISE, or DOE. This research was also supported in part by the National Nuclear Security Administration (NNSA) Advanced Simulation and Computing (ASC) program, Advanced Technology Development and Mitigation (ATDM) subprogram. This work was performed under the auspices of the National Nuclear Security Administration of the US Department of Energy at Los Alamos National Laboratory under Contract No. DE-AC52-06NA25396. The authors gratefully acknowledge the support of the US Department of Energy National Nuclear Security Administration Advanced Simulation and Computing Program. We thank the anonymous reviewers for their time and efforts to provide constructive feedback to initial manuscripts. Their comments helped to improve and clarify points made in this paper.

## ORCID iDs

Pake Melland  <https://orcid.org/0000-0003-1980-9072>

Jason Albright  <https://orcid.org/0000-0002-4099-8990>

Nathan M Urban  <https://orcid.org/0000-0002-2264-3512>

## References

- [1] LeVeque R J 1992 *Numerical Methods for Conservation Laws* vol 132 (Berlin: Springer)
- [2] Toro E F 2013 *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction* (Berlin: Springer Science & Business Media)
- [3] VonNeumann J and Richtmyer R D 1950 *J. Appl. Phys.* **21** 232–7
- [4] Wilkins M L 1980 *J. Comput. Phys.* **36** 281–303
- [5] Caramana E, Burton D, Shashkov M J and Whalen P 1998 *J. Comput. Phys.* **146** 227–62
- [6] Noh W F 1987 *J. Comput. Phys.* **72** 78–120
- [7] Urban N, Shashkov M and Albright J 2019 Machine learning-based optimization strategies for artificial viscosity part II *Machine Learning for Computational Fluid and Solid Dynamics (Santa Fe, NM, 19–21 February 2019)* (available at: <https://permalink.lanl.gov/object/view?what=info:lanl-repo/lareport/LA-UR-19-21431>)
- [8] Rackauckas C, Ma Y, Martensen J, Warner C, Zubov K, Supekar R, Skinner D and Ramadhan A 2020 (arXiv:2001.04385)
- [9] Tompson J, Schlachter K, Sprechmann P and Perlin K 2017 Accelerating Eulerian fluid simulation with convolutional networks *Proc. 34th Int. Conf. on Machine Learning* vol 70 (JMLR. org) pp 3424–33 (<https://proceedings.mlr.press/v70/tompson17a.html>)
- [10] Kim B, Azevedo V C, Thuerey N, Kim T, Gross M and Solenthaler B 2019 *Computer Graphics Forum* **38** 59–70
- [11] Sirignano J and Spiliopoulos K 2018 *J. Comput. Phys.* **375** 1339–64
- [12] Han J, Jentzen A and Weinan E 2018 *Proc. Natl Acad. Sci.* **115** 8505–10
- [13] de Bezenac E, Pajot A and Gallinari P 2017 (arXiv:1711.07970)
- [14] Zhu Y, Zabarav N, Koutsourelakis P S and Perdikaris P 2019 *J. Comput. Phys.* **394** 56–81
- [15] Bar-Sinai Y, Hoyer S, Hickey J and Brenner M P 2019 *Proc. Natl Acad. Sci.* **116** 15344–9
- [16] Mishra S 2018 (arXiv:1807.09519)
- [17] Maulik R, San O, Rasheed A and Vedula P 2019 *J. Fluid Mech.* **858** 122–44
- [18] Mohan A T, Lubbers N, Livescu D and Chertkov M 2020 *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations* Embedding hard physical constraints in convolutional neural networks for 3d turbulence (<https://openreview.net/forum?id=IaXBtMNFaa>)
- [19] Rasp S, Pritchard M S and Gentine P 2018 *Proc. Natl Acad. Sci.* **115** 9684–9
- [20] Brenowitz N D and Bretherton C S 2018 *Geophys. Res. Lett.* **45** 6289–98
- [21] O’Gorman P A and Dwyer J G 2018 *J. Adv. Model. Earth Syst.* **10** 2548–63
- [22] Large W G, McWilliams J C and Doney S C 1994 *Rev. Geophys.* **32** 363–403
- [23] Hourdin F et al 2017 *Bull. Am. Meteorol. Soc.* **98** 589–602
- [24] Schmidt G A, Bader D, Donner L J, Elsaesser G S, Golaz J C, Hannay C, Molod A, Neale R and Saha S 2017 *Geosci. Model Dev.* **10** 3207
- [25] Santner T J, Williams B J, Notz W and Williams B J 2003 *The Design and Analysis of Computer Experiments* vol 1 (Berlin: Springer)
- [26] Pronzato L and Müller W G 2012 *Stat. Comput.* **22** 681–701
- [27] Rumelhart D E, Hinton G E and Williams R J 1986 *Nature* **323** 533–6
- [28] Hecht-Nielsen R 1992 Theory of the backpropagation neural network *Neural Networks for Perception* (Amsterdam: Elsevier) pp 65–93
- [29] Abadi M et al 2016 Tensorflow: a system for large-scale machine learning *12th USENIX Symp. on Operating Systems Design and Implementation (OSDI 16)* pp 265–83
- [30] Paszke A et al 2019 Pytorch: an imperative style, high-performance deep learning library *Advances in Neural Information Processing Systems 32* eds H Wallach, H Larochelle, A Beygelzimer, F d’Alché-Buc, E Fox and R Garnett (Curran Associates, Inc.) pp 8026–37
- [31] Baydin A G, Pearlmutter B A, Radul A A and Siskind J M 2018 *J. Mach. Learn. Res.* **18** 1–43 (<https://jmlr.org/papers/v18/17-468.html>)

- [32] Horwedel J, Worley B, Oblow E, Pin F and Wright R 1988 Gress (gradient enhanced software system) version 0. 0 user's manual *Technical Report* (Oak Ridge National Laboratory (ORNL))
- [33] Kubota K and Iri M 1990 PADRE2, version 1—user's manual *Research memorandum RMI 90-01*
- [34] Innes M, Edelman A, Fischer K, Rackauckas C, Saba E, Shah V B and Tebbutt W 2019 (arXiv:1907.07587)
- [35] Wang F, Decker J, Wu X, Essertel G and Rompf T 2018 Backpropagation with callbacks: foundations for efficient and expressive differentiable programming *Adv. Neural Inform. Process. Syst.* **31** 10180–91
- [36] Innes M 2018 *J. Open Source Softw.* **3** 602
- [37] Innes M, Saba E, Fischer K, Gandhi D, Rudilosso M C, Joy N M, Karmali T, Singh A P and Shah V 2018 (arXiv:1811.01457)
- [38] Gunzburger M 2000 *Flow Turbul. Combust.* **65** 249–72
- [39] Asch M, Bocquet M and Nodet M 2016 *Data Assimilation: Methods, Algorithms and Applications* (Philadelphia, PA: SIAM)
- [40] Sod G A 1977 *J. Fluid Mech.* **83** 785–94
- [41] Albright J and Shashkov M 2020 *Comput. Fluids* **104580**
- [42] Sod G A 1978 *J. Comput. Phys.* **27** 1–31
- [43] Caramana E J, Shashkov M J and Whalen P P 1998 *J. Comput. Phys.* **144** 70–97
- [44] Cybenko G 1989 *Math. Control Signals Syst.* **2** 303–14
- [45] Strang G 2019 *Linear Algebra and Learning From Data* (Wellesley, MA: Wellesley-Cambridge Press)
- [46] Loubère R 2005 Validation test case suite for compressible hydrodynamics computation *Technical Report* (Los Alamos, NM: Los Alamos National Laboratory) (available at: [http://loubere.free.fr/images/test\\_suite.PDF](http://loubere.free.fr/images/test_suite.PDF))
- [47] Kingma D P and Ba J 2014 (arXiv:1412.6980)
- [48] Bezanson J, Edelman A, Karpinski S and Shah V B 2017 *SIAM Rev.* **59** 65–98
- [49] Rackauckas C, Ma Y, Dixit V, Guo X, Innes M, Revels J, Nyberg J and Ivaturi V 2018 (arXiv:1812.01892)
- [50] Rackauckas C, Innes M, Ma Y, Bettencourt J, White L and Dixit V 2019 (arXiv:1902.02376)